Case Study: Collecting Job Data Using APIs

Objectives

After completing this project we will be able to:

- Collect job data using Jobs API.
- Store the collected data into an excel spreadsheet.

> **Note: Before starting with the assignment make sure to read all the instructions and then move ahead with the coding part.**

Instructions

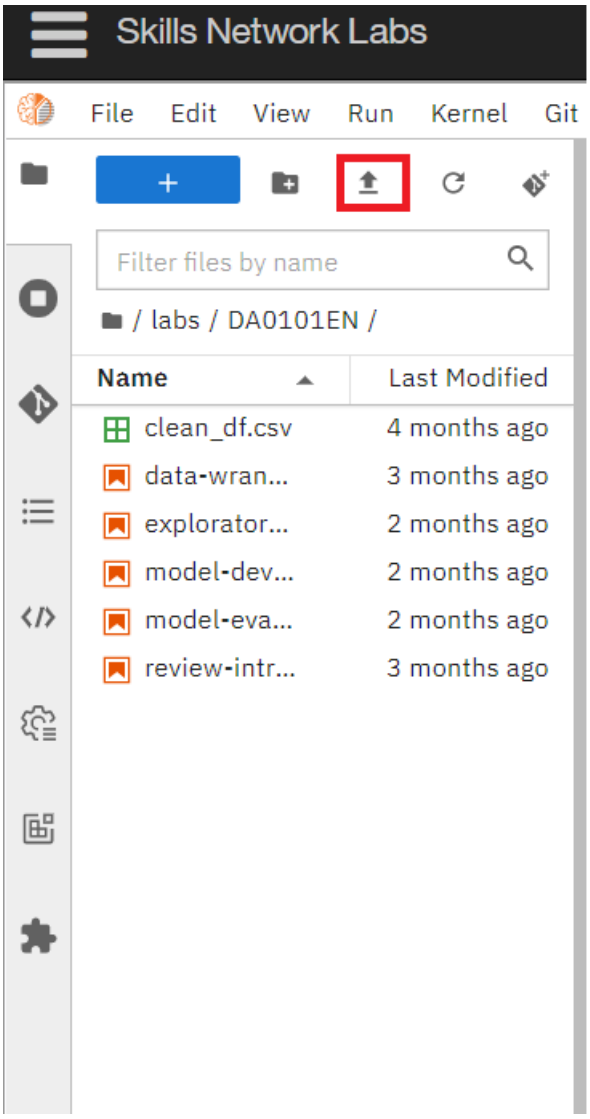
To run the actual lab, firstly you need to click on the Jobs_API notebook link. The file contains flask code which is required to run the Jobs API data.

Now, to run the code in the file that opens up follow the below steps.

Step1: Download the file.

Step2: Upload the file into your current Jupyter environment using the upload button in your Jupyter interface. Ensure that the file is in the same folder as your working .ipynb file.

Step 2: If working in a local Jupyter environment, use the "Upload" button in your Jupyter interface to upload the Jobs_API notebook into the same folder as your current .ipynb file.



Step3: Open the Jobs_API notebook, and run all the cells to start the Flask application. Once the server is running, you can access the API from the URL provided in the notebook.

If you want to learn more about flask, which is optional, you can click on this link here.

Once you run the flask code, you can start with your assignment.

Dataset Used in this Assignment

The dataset used in this lab comes from the following source: https://www.kaggle.com/promptcloud/jobs-on-naukricom under the under a **Public Domain license**.

> Note: We are using a modified subset of that dataset for the lab, so to follow the lab instructions successfully please use the dataset provided with the lab, rather than the dataset from the original source.

The original dataset is a csv. We have converted the csv to json as per the requirement of the lab.

Warm-Up Exercise

Before you attempt the actual lab, here is a fully solved warmup exercise that will help you to learn how to access an API.

Using an API, let us find out who currently are on the International Space Station (ISS).
The API at http://api.open-notify.org/astros.json gives us the information of astronauts currently on ISS in json format.
You can read more about this API at http://open-notify.org/Open-Notify-API/People-In-Space/

In [1]:
```
import requests # you need this module to make an API call
import pandas as pd
```
In [3]:
```
api_url = "http://api.open-notify.org/astros.json" # this url gives use the astronaut data
```
In [5]:
```
response = requests.get(api_url) # Call the API using the get method and store the
                                 # output of the API call in a variable called response.
```
In [7]:
```
if response.ok:            # if all is well() no errors, no network timeouts)
    data = response.json()  # store the result in json format in a variable called data
                            # the variable data is of type dictionary.
```
In [9]:
```
print(data)   # print the data just to check the output or for debugging
```
```
{'people': [{'craft': 'ISS', 'name': 'Oleg Kononenko'}, {'craft': 'ISS', 'name': 'Nikolai Chub'}, {'craft': 'ISS', 'name': 'Tracy Caldwell Dyson'}, {'craft': 'ISS', 'name': 'Matthew Dominick'}, {'craft': 'ISS', 'name': 'Michael Barratt'}, {'craft': 'ISS', 'name': 'Jeanette Epps'}, {'craft': 'ISS', 'name': 'Alexander Grebenkin'}, {'craft': 'ISS', 'name': 'Butch Wilmore'}, {'craft': 'ISS', 'name': 'Sunita Williams'}, {'craft': 'Tiangong', 'name': 'Li Guangsu'}, {'craft': 'Tiangong', 'name': 'Li Cong'}, {'craft': 'Tiangong', 'name': 'Ye Guangfu'}], 'number': 12, 'message': 'success'}
```
Print the number of astronauts currently on ISS.

In [12]:
```
print(data.get('number'))
```
```
12
```
Print the names of the astronauts currently on ISS.

In [20]:
```
astronauts = data.get('people')
print("There are {} astronauts on ISS".format(len(astronauts)))
print("And their names are :")
for astronaut in astronauts:
    print(astronaut.get('name'))
```
```
There are 12 astronauts on ISS
And their names are :
Oleg Kononenko
Nikolai Chub
Tracy Caldwell Dyson
Matthew Dominick
Michael Barratt
Jeanette Epps
Alexander Grebenkin
Butch Wilmore
Sunita Williams
Li Guangsu
Li Cong
Ye Guangfu
```
Hope the warmup was helpful. Good luck with your next lab!

Lab: Collect Jobs Data using Jobs API

Objective: Determine the number of jobs currently open for various technologies and for various locations

Collect the number of job postings for the following locations using the API:

- Los Angeles
- New York
- San Francisco
- Washington DC
- Seattle
- Austin
- Detroit

In [26]:
```
#Import required libraries
import pandas as pd
import json
```
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%20Data%20Using%20APIs/jobs.json#### Write a function to get the number of jobs for the Python technology.

> Note: While using the lab you need to pass the **payload** information for the **params** attribute in the form of **key value** pairs.

Refer the ungraded **rest api lab** in the course **Python for Data Science, AI & Development** link

The keys in the json are

- Job Title
- Job Experience Required
- Key Skills
- Role Category
- Location
- Functional Area
- Industry
- Role

You can also view the json file contents from the following json URL.

In [ ]:

(TASK 1:) Function to Get the Number of Jobs for the Python Technology.

In [31]:

```
import requests
import json

# API URL containing job postings
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

def get_number_of_jobs_T(technology):
    """
    This function takes a technology name as input (e.g., 'Python')
    and returns the number of job postings related to that technology.
    """

    # Step 1: Fetch the job data from the API
    response = requests.get(api_url)  # Send a GET request to the API
    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    # Step 2: Convert the response JSON data into a Python dictionary
    jobs_data = response.json()

    # Step 3: Initialize a counter for job postings related to the given technology
    number_of_jobs = 0

    # Step 4: Loop through the job postings and count jobs mentioning the technology
    for job in jobs_data:
        if 'Key Skills' in job and technology.lower() in job['Key Skills'].lower():
            number_of_jobs += 1  # Increase count if technology is found in job skills

    # Step 5: Return the result as a tuple (technology, number_of_jobs)
    return technology, number_of_jobs

# Example usage:
tech = "Python"
result = get_number_of_jobs_T(tech)
print(f"Number of jobs for {result[0]}: {result[1]}")
```

```
Number of jobs for Python: 1173
```

Calling the function for Python and checking if it works.

Job Listings by Technology: Python

This function call searches the dataset for job listings containing the skill **"Python"**.

Result: It returns the total number of job listings that mention Python in their “Key Skills.”

Useful for identifying general demand for Python in the job market.

In [29]:

```
get_number_of_jobs_T("Python")
```

Out[29]:

```
('Python', 1173)
```

Code Explanation (Layman's Terms)

Import required libraries:

- requests for making API calls
- json for handling JSON data

Define API URL:

- The api_url contains job postings in JSON format.

Create get_number_of_jobs_T(technology) function:

- Fetch job data from the API using requests.get().
- Convert JSON response to a Python dictionary using .json().
- Initialize number_of_jobs = 0 to keep track of job counts.
- Loop through all job postings and check if the technology name appears in "Key Skills".
- If found, increase the count.
- Return the technology name and job count as a tuple.

Call the function for 'Python' jobs:

- Print the technology name and job count.

In [ ]:

(TASK 2:) Extended Code: Getting Number of Jobs for the Python for Many Different Locations

The below updated version of the function allows us to check job postings for a specific technology in multiple locations.

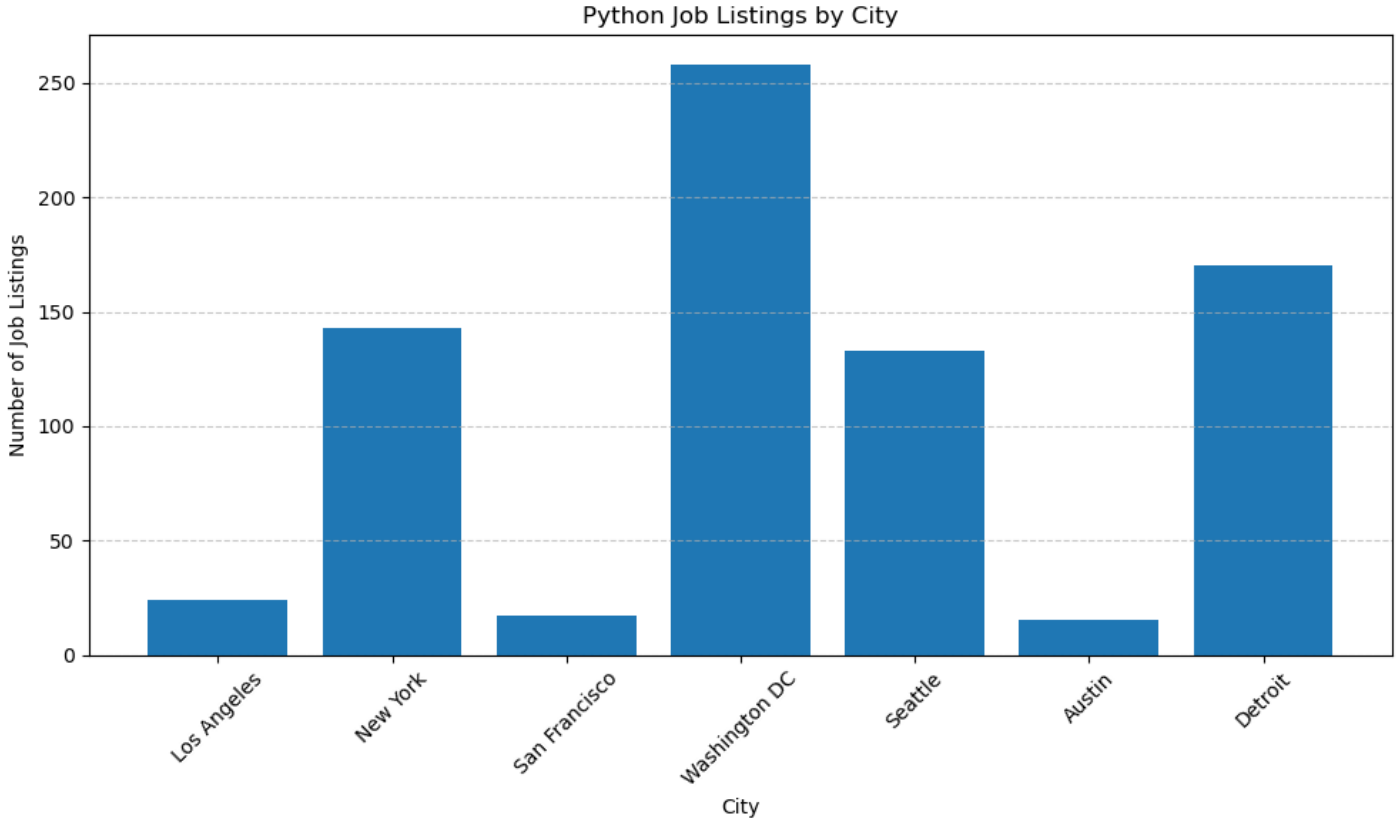
In [35]:

```
import requests
import json

# API URL containing job postings
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

def get_number_of_jobs_T_L(technology, locations):
    """
    This function takes a technology name (e.g., 'Python') and a list of locations.
    It returns the number of job postings related to that technology in each location.
    """

    # Step 1: Fetch the job data from the API
    response = requests.get(api_url)  # Send a GET request to the API
    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    # Step 2: Convert the response JSON data into a Python dictionary
    jobs_data = response.json()

    # Step 3: Initialize a dictionary to store job counts for each location
    job_counts = {location: 0 for location in locations}

    # Step 4: Loop through the job postings and count jobs mentioning the technology in the given locations
    for job in jobs_data:
        if 'Key Skills' in job and technology.lower() in job['Key Skills'].lower():
            for location in locations:
                if 'Location' in job and location.lower() in job['Location'].lower():
                    job_counts[location] += 1  # Increase count for the location

    # Step 5: Return the results as a dictionary
    return job_counts

# Example Usage:
locations_list = ["Los Angeles", "New York", "San Francisco", "Washington DC", "Seattle", "Austin", "Detroit"]
technology = "Python"

job_results = get_number_of_jobs_T_L(technology, locations_list)

# Print results
print(f"Number of {technology} jobs by location:")
for location, count in job_results.items():
    print(f"{location}: {count} jobs")
```

```
Number of Python jobs by location:
Los Angeles: 24 jobs
New York: 143 jobs
San Francisco: 17 jobs
Washington DC: 258 jobs
Seattle: 133 jobs
Austin: 15 jobs
Detroit: 170 jobs
```

In [93]:

```
# Re-import required packages after environment reset
import matplotlib.pyplot as plt

# Job count data
cities = ["Los Angeles", "New York", "San Francisco", "Washington DC", "Seattle", "Austin", "Detroit"]
job_counts = [24, 143, 17, 258, 133, 15, 170]

# Create bar chart
plt.figure(figsize=(10, 6))
plt.bar(cities, job_counts)
plt.title("Python Job Listings by City")
plt.xlabel("City")
plt.ylabel("Number of Job Listings")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Save figure
chart_path = "python_job_listings_by_city_chart.png"
plt.savefig(chart_path)
plt.show()

print(f"Python Job Listings by City bar chart saved as '{chart_path}' – ready for your PowerPoint slide!")
```




```
Python Job Listings by City bar chart saved as 'python_job_listings_by_city_chart.png' – ready for your PowerPoint slide!
```

Python Job Listings by City

This bar chart visualizes the number of job postings mentioning **Python** across major U.S. cities:

Key Insights

1. **Washington DC** leads with the highest demand for Python roles with 258 listings, likely due to government and federal tech-related work.
2. **Detroit** and **New York** follow, with 170 and 143 listings respectively, reflecting growing demand in industrial and finance sectors.
3. **Seattle** shows solid demand (133 jobs), which aligns with expectations as a major tech hub.
4. **Austin**, **San Francisco**, and **Los Angeles** show fewer postings in this dataset — potentially due to sampling scope or sector focus.

Python continues to be a high-demand skill across diverse metro areas, especially where data-driven roles are concentrated.

Job Listings by Technology Across Multiple Cities

This function searches for job listings mentioning **a specific technology** (e.g., "Python") across a **list of cities**.

It helps us understand **where demand for a particular tech skill is highest geographically**.

Result Output:

```
Number of Python jobs by location:
Los Angeles: 24 jobs
New York: 143 jobs
...

Ideal for identifying city-specific demand for a given skill — helpful for location-based job targeting.

Helps provide a sense of general job market activity in each city.

This can later be turned into a bar chart to visually compare job density by city — useful for relocation
or hiring strategy.
```

Code Explanation (Layman's Terms)

Import necessary libraries (requests & json).

Define API URL containing job postings.

Create get_number_of_jobs_T_L(technology, locations) function:

- Fetch job data from the API.
- Convert response into a Python dictionary.
- Create a dictionary (job_counts) to store job counts for each location.

Loop through job postings:

- Check if the technology appears in Key Skills.
- Check if the job Location matches any location in the list.
- Increase the job count for matching locations.
- Return a dictionary with job counts for each location.

Call the function for 'Python' jobs in multiple locations.

Print results in a readable format.

(TASK 3:) Write a function to find number of jobs in US for a location of your choice (eg.Los Angeles)

In [13]:

```
import requests
import json

# API URL containing job postings
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

def get_number_of_jobs_L(location):
    """
    This function takes a location (e.g., 'Los Angeles') as input.
    It returns the number of job postings available in that location.
    """

    # Step 1: Fetch job data from the API
    response = requests.get(api_url)  # Send a GET request to the API
    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    # Step 2: Convert the response JSON data into a Python dictionary
    jobs_data = response.json()

    # Step 3: Initialize job count for the given location
    number_of_jobs = 0

    # Step 4: Loop through job postings and count jobs in the specified location
    for job in jobs_data:
        if 'Location' in job and location.lower() in job['Location'].lower():
            number_of_jobs += 1  # Increase count if location matches

    # Step 5: Return the result as a tuple
    return location, number_of_jobs

# Example Usage: Find number of jobs in Los Angeles
location = "Los Angeles"
job_count = get_number_of_jobs_L(location)

# Print the result
print(f"Number of jobs in {job_count[0]}: {job_count[1]}")
```

```
Number of jobs in Los Angeles: 640
```

Call the function for Los Angeles and check if it is working.

In [15]:

```
get_number_of_jobs_L("Los Angeles")
```

Out[15]:

```
('Los Angeles', 640)
```

In [17]:

```
get_number_of_jobs_L(location)
```

Out[17]:

```
('Los Angeles', 640)
```

Code Explanation (Layman's Terms)

Import necessary libraries (requests & json).

Define API URL containing job postings.

Create get_number_of_jobs_T_L(technology, locations) function:

- Fetch job data from the API.
- Convert response into a Python dictionary.
- Set an initial job count to 0.

Loop through job postings:

- Check if the job Location matches the input location.
- Increase the job count for that location.
- Return a tuple with the location name and job count.

Call the function for "Los Angeles" and print results.

In [ ]:

(TASK 4:) Extended Code: Function to Get Job Counts for Multiple Locations

This updated function allows you to check job counts for multiple cities at the same time.

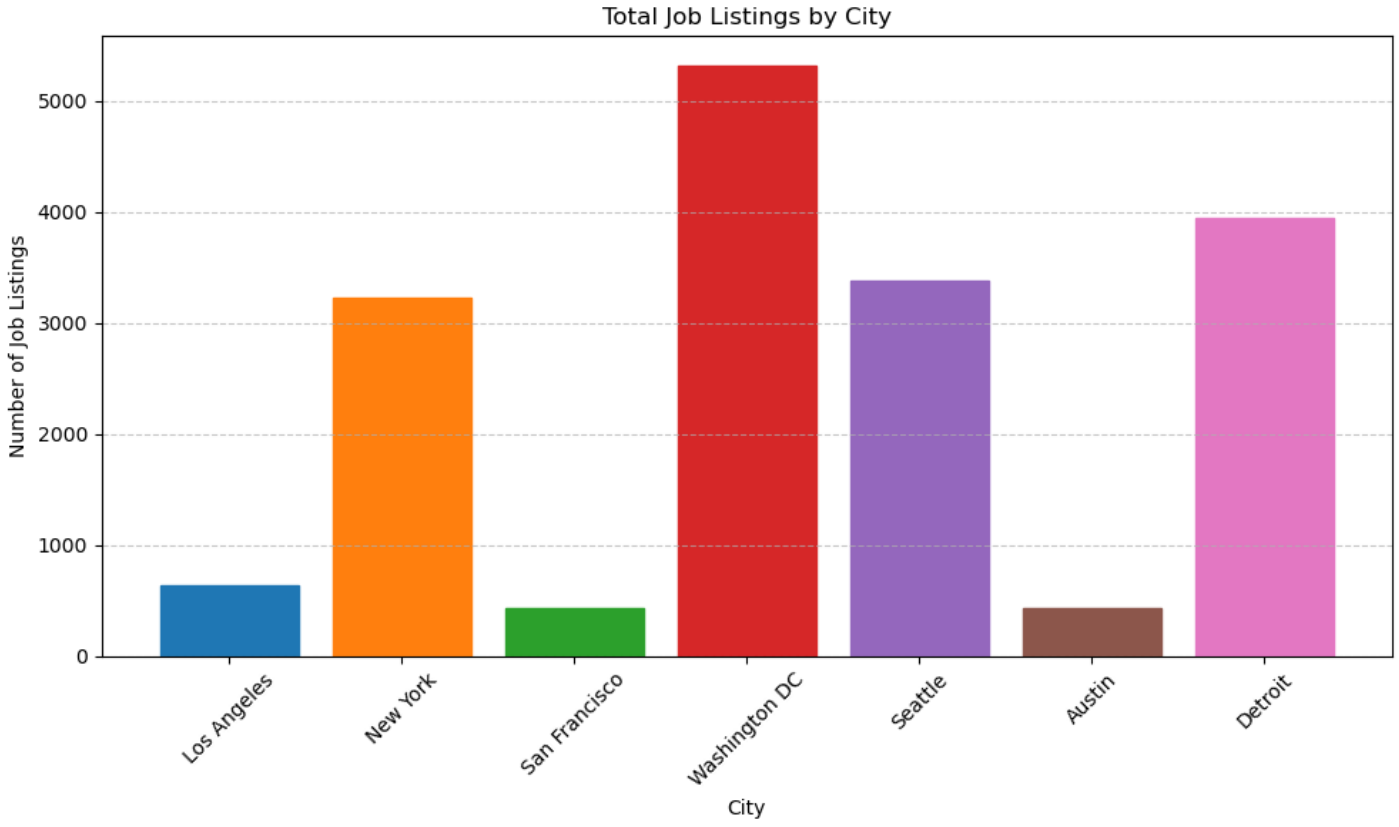
In [18]:

```
import requests
import json

# API URL containing job postings
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

def get_number_of_jobs_multiple_locations(locations):
    """
    This function takes a list of locations as input.
    It returns a dictionary containing the number of job postings for each location.
    """

    # Step 1: Fetch job data from the API
    response = requests.get(api_url)
    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    # Step 2: Convert the response JSON data into a Python dictionary
    jobs_data = response.json()

    # Step 3: Initialize a dictionary to store job counts for each location
    job_counts = {location: 0 for location in locations}

    # Step 4: Loop through job postings and count jobs for each location
    for job in jobs_data:
        if 'Location' in job:
            job_location = job['Location'].lower()
            for location in locations:
                if location.lower() in job_location:
                    job_counts[location] += 1

    # Step 5: Return the job counts dictionary
    return job_counts

# Example Usage: Find number of jobs in multiple cities
locations = ["Los Angeles", "New York", "San Francisco", "Washington DC", "Seattle", "Austin", "Detroit"]
job_counts = get_number_of_jobs_multiple_locations(locations)

# Print results
for city, count in job_counts.items():
    print(f"Number of jobs in {city}: {count}")
```

```
Number of jobs in Los Angeles: 640
Number of jobs in New York: 3226
Number of jobs in San Francisco: 435
Number of jobs in Washington DC: 5316
Number of jobs in Seattle: 3375
Number of jobs in Austin: 434
Number of jobs in Detroit: 3945
```

In [91]:

```
# Re-import necessary package after environment reset
import matplotlib.pyplot as plt

# Updated job count data (total jobs by city)
cities_updated = ["Los Angeles", "New York", "San Francisco", "Washington DC", "Seattle", "Austin", "Detroit"]
job_counts_updated = [640, 3226, 435, 5316, 3375, 434, 3945]

# Create colorful bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(cities_updated, job_counts_updated)

# Color each bar differently
for i, bar in enumerate(bars):
    bar.set_color(plt.cm.tab10(i % 10))

# Add chart details
plt.title("Total Job Listings by City")
plt.xlabel("City")
plt.ylabel("Number of Job Listings")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Save figure
chart_path = "total_job_listings_by_city_chart.png"
plt.savefig(chart_path)
plt.show()

print(f"Total Job Listings by City bar chart saved as '{chart_path}' – ready for your PowerPoint slide!")
```



Total Job Listings by City bar chart saved as 'total_job_listings_by_city_chart.png' – ready for your PowerPoint slide!

Total Job Listings by City (All Skills)

This bar chart displays the number of total job listings across different U.S. cities:

Key Insights

1. **Washington DC** has the largest number of job postings, with **5316** overall, highlighting it as a major employment hub.
2. **Detroit**, **Seattle**, and **New York** with **3945, 3375, and 3226**, respectively, also show strong job markets with thousands of openings.
3. **Los Angeles**, **San Francisco**, and **Austin** have notably fewer listings compared to top cities, which may be due to different economic focuses or sample data limitations.

The data shows that larger urban centers and government-heavy regions have a higher volume of job opportunities across all sectors.

Code Explanation (Layman's Terms)

Import necessary libraries (requests & json).

Define API URL containing job postings.

Creates get_number_of_jobs_multiple_locations(locations) function:

- Accepts a list of locations.
- Fetches job postings from the API.
- Converts the response into a Python dictionary.
- Initializes a dictionary (job_counts) to store job counts for each location.

Loop through job postings:

- Checks if a location in the list is present in the job listing.
- Updates the job count for each matched location.
- Returns the dictionary containing job counts.

Calls the function for multiple cities and prints results.

In [ ]:

(TASK 5:) Store the results in an excel file.

Call the API for all the given technologies above and write the results in an excel spreadsheet.

If you do not know how create excel file using python, double click here for **hints**.

Create a python list of all technologies for which you need to find the number of jobs postings.

In [22]:
!pip install openpyxl

```
Requirement already satisfied: openpyxl in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (3.1.3)
Requirement already satisfied: et-xmlfile in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from openpyxl) (1.1.0)
```

In [23]:
import requests
import pandas **as** pd
from openpyxl **import** Workbook
Import libraries required to create excel spreadsheet

In [24]:

```
# **STEP 1: Define List of Technologies**
technologies = ["Python", "Java", "JavaScript", "C++", "Ruby", "Swift", "Go", "Kotlin"]

# **STEP 2: Define API URL**
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# **STEP 3: Create a Function to Get Job Counts for Each Technology**
def get_number_of_jobs_T(technology):
    """
    Function to fetch job postings count for a given technology.
    """
    response = requests.get(api_url)

    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    jobs_data = response.json()  # Convert JSON response into Python Dictionary
    job_count = sum(1 for job in jobs_data if 'Key Skills' in job and technology.lower() in job['Key Skills'].lower())

    return technology, job_count
```

Create a workbook and select the active worksheet

In [25]:

```
# **STEP 4: Create a Workbook and Select the Active Worksheet**
wb = Workbook()
ws = wb.active
ws.title = "Job Postings"

# **STEP 5: Write Header Row in Excel File**
ws.append(["Technology", "Job Postings"])  # Column Titles
```

Find the number of jobs postings for each of the technology in the above list. Write the technology name and the number of jobs postings into the excel spreadsheet.

In [26]:

```
# **STEP 6: Find Job Postings for Each Technology and Write to Excel**
job_data = []

for tech in technologies:
    tech_name, job_count = get_number_of_jobs_T(tech)
    job_data.append([tech_name, job_count])  # Store data in list
    ws.append([tech_name, job_count])  # Append data to Excel sheet
```

Save into an excel spreadsheet named **job-postings.xlsx**.

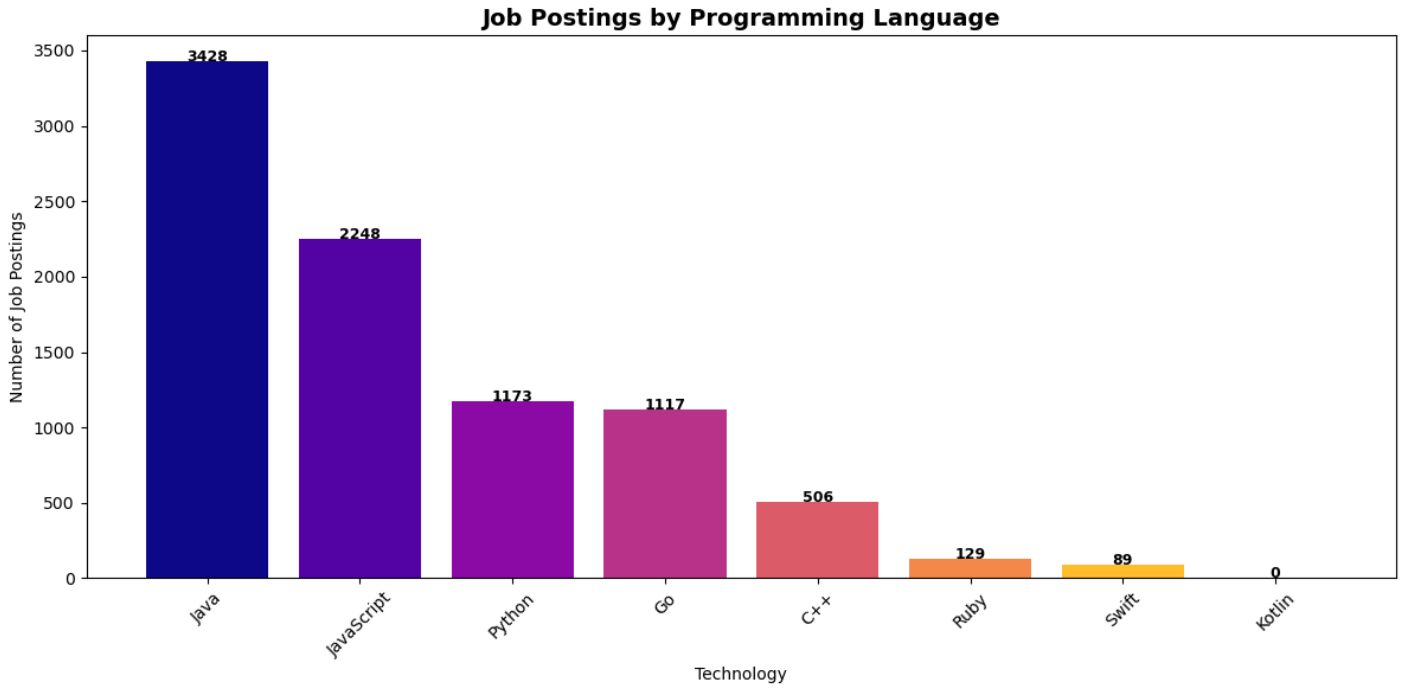
In [27]:

```
# **STEP 7: Save Data into an Excel Spreadsheet**
excel_filename = "job-postings.xlsx"
wb.save(excel_filename)

# **STEP 8: Convert Data to DataFrame and Display Results**
df = pd.DataFrame(job_data, columns=["Technology", "Job Postings"])
print(df)

# **STEP 9: Confirm Data Saved Successfully**
print(f"Job postings data has been successfully saved to '{excel_filename}'!")
```

```
   Technology  Job Postings
0      Python          1173
1        Java          3428
2  JavaScript          2248
3         C++           506
4        Ruby           129
5       Swift            89
6          Go          1117
7      Kotlin             0
Job postings data has been successfully saved to 'job-postings.xlsx'!
```

In [58]:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# STEP 1: Define List of Technologies
technologies = ["Python", "Java", "JavaScript", "C++", "Ruby", "Swift", "Go", "Kotlin"]

# STEP 2: Define API URL
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# STEP 3: Function to Get Job Counts
def get_number_of_jobs_T(technology):
    response = requests.get(api_url)
    if response.status_code != 200:
        return technology, 0
    jobs_data = response.json()
    job_count = sum(1 for job in jobs_data if 'Key Skills' in job and technology.lower() in job['Key Skills'].lower())
    return technology, job_count

# STEP 4: Collect Job Data
job_results = [get_number_of_jobs_T(tech) for tech in technologies]
df = pd.DataFrame(job_results, columns=["Technology", "Job Postings"]).sort_values(by="Job Postings", ascending=False)

# STEP 5: Visualize Results
plt.figure(figsize=(12, 6))
colors = plt.cm.plasma(np.linspace(0, 1, len(df)))
bars = plt.bar(df["Technology"], df["Job Postings"], color=colors)

# Add labels
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 1, str(height), ha='center', fontsize=9, fontweight='bold')

plt.title("Job Postings by Programming Language", fontsize=14, fontweight='bold')
plt.xlabel("Technology")
plt.ylabel("Number of Job Postings")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("tech_job_postings_chart4.png", dpi=300)
plt.show()
```



In [ ]:

In [20]:

```
!pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.3-py2.py3-none-any.whl (251 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.3/251.3 kB 26.7 MB/s eta 0:00:00
Collecting et-xmlfile (from openpyxl)
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.3
```

In [28]:

```
import requests
import pandas as pd
from openpyxl import Workbook

# **STEP 1: Define List of Technologies**
technologies = ["Python", "Java", "JavaScript", "C++", "Ruby", "Swift", "Go", "Kotlin"]

# **STEP 2: Define API URL**
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# **STEP 3: Create a Function to Get Job Counts for Each Technology**
def get_number_of_jobs_T(technology):
    """
    Function to fetch job postings count for a given technology.
    """
    response = requests.get(api_url)

    if response.status_code != 200:
        return f"Error: Unable to fetch data. Status Code: {response.status_code}"

    jobs_data = response.json()  # Convert JSON response into Python Dictionary
    job_count = sum(1 for job in jobs_data if 'Key Skills' in job and technology.lower() in job['Key Skills'].lower())

    return technology, job_count

# **STEP 4: Create a Workbook and Select the Active Worksheet**
wb = Workbook()
ws = wb.active
ws.title = "Job Postings"

# **STEP 5: Write Header Row in Excel File**
ws.append(["Technology", "Job Postings"])  # Column Titles

# **STEP 6: Find Job Postings for Each Technology and Write to Excel**
job_data = []

for tech in technologies:
    tech_name, job_count = get_number_of_jobs_T(tech)
    job_data.append([tech_name, job_count])  # Store data in list
    ws.append([tech_name, job_count])  # Append data to Excel sheet

# **STEP 7: Save Data into an Excel Spreadsheet**
excel_filename = "job-postings.xlsx"
wb.save(excel_filename)

# **STEP 8: Convert Data to DataFrame and Display Results**
df = pd.DataFrame(job_data, columns=["Technology", "Job Postings"])
print(df)

# **STEP 9: Confirm Data Saved Successfully**
print(f"Job postings data has been successfully saved to '{excel_filename}'!")
```

```
   Technology  Job Postings
0      Python          1173
1        Java          3428
2  JavaScript          2248
3         C++           506
4        Ruby           129
5       Swift            89
6          Go          1117
7      Kotlin             0
Job postings data has been successfully saved to 'job-postings.xlsx'!
```

Step	Description
1. Create List of Technologies	Define a list of technologies for which job postings need to be retrieved.
2. Import Required Libraries	Import `requests` for API calls, `pandas` for data processing, and `openpyxl` for Excel handling.
3. Create Workbook & Worksheet	Initialize an Excel workbook and set an active worksheet for storing results.
4. Fetch Job Postings from API	Call the API for each technology, extract job postings count, and store the results.
5. Write Data to Excel	Write the technology names and job postings count into the Excel sheet.
6. Save Excel File	Save the file as `job-postings.xlsx` and verify the data.

Step	Description
1. Create List of Technologies	Define a list of technologies for which job postings need to be retrieved.
2. Import Required Libraries	Import `requests` for API calls, `pandas` for data processing, and `openpyxl` for Excel handling.
3. Create Workbook & Worksheet	Initialize an Excel workbook and set an active worksheet for storing results.
4. Fetch Job Postings from API	Call the API for each technology, extract job postings count, and store the results.
5. Write Data to Excel	Write the technology names and job postings count into the Excel sheet.
6. Save Excel File	Save the file as `job-postings.xlsx` and verify the data.

(TASK 6:) In the similar way, you can try for below given technologies and results can be stored in an excel sheet.

Collect the number of job postings for the following languages using the API:

- C
- C#
- C++
- Java
- JavaScript
- Python
- Scala
- Oracle
- SQL Server
- MySQL Server
- PostgreSQL
- MongoDB

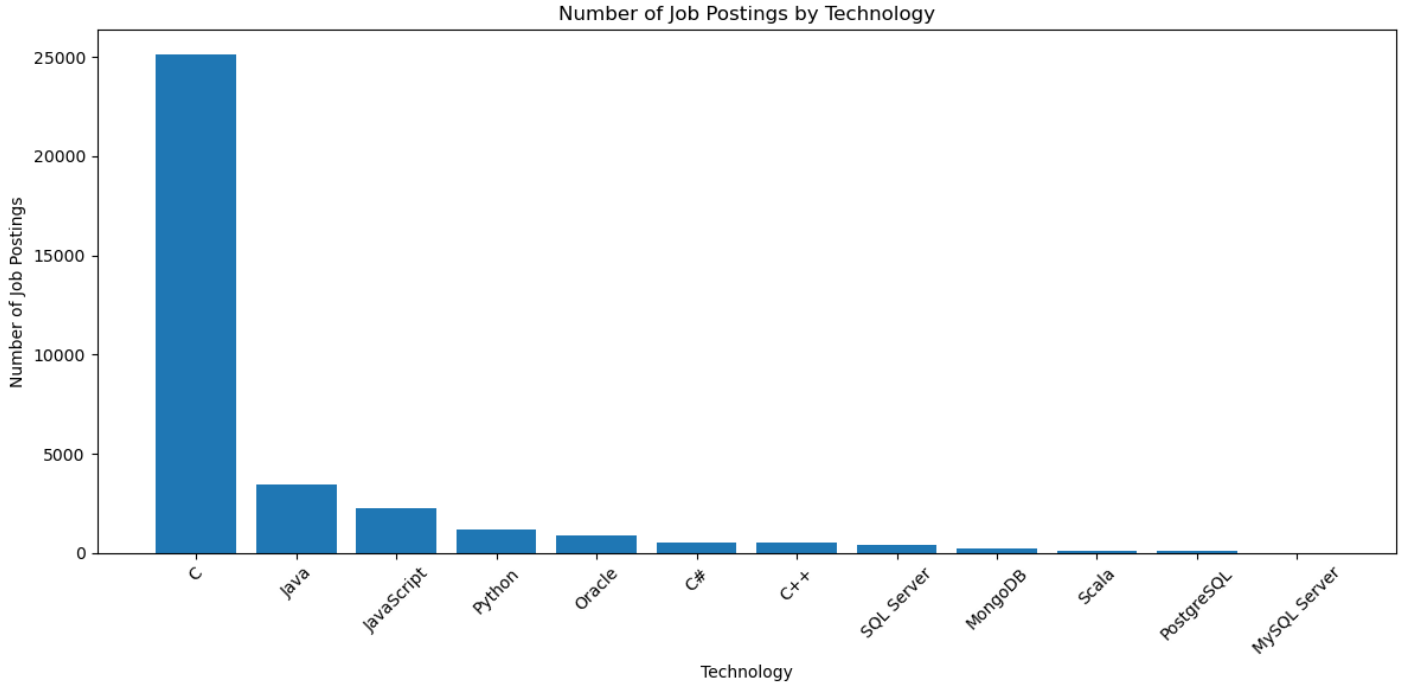
In [32]:

```
import requests
import pandas as pd
from openpyxl import Workbook

# **STEP 1: Define List of Technologies**
technologies = [
    "C", "C#", "C++", "Java", "JavaScript", "Python",
    "Scala", "Oracle", "SQL Server", "MySQL Server", "PostgreSQL", "MongoDB"
]

# **STEP 2: Define API Endpoint**
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# **STEP 3: Create a DataFrame to Store Results**
job_results = []

# **STEP 4: Function to Fetch Job Count for Each Technology**
def get_number_of_jobs(technology):
    response = requests.get(api_url)
    if response.status_code == 200:
        job_data = response.json()
        # Count the number of job postings containing the technology in "Key Skills"
        job_count = sum(1 for job in job_data if technology.lower() in job.get("Key Skills", "").lower())
        return job_count
    else:
        return None  # Return None if API fails

# **STEP 5: Fetch Jobs for Each Technology**
for tech in technologies:
    num_jobs = get_number_of_jobs(tech)
    if num_jobs is not None:
        job_results.append({"Technology": tech, "Job Postings": num_jobs})
    else:
        job_results.append({"Technology": tech, "Job Postings": "API Error"})

# **STEP 6: Convert Data to DataFrame**
df_jobs = pd.DataFrame(job_results)

# **STEP 7: Print the Results in a Table Format**
print("\nJob Postings by Technology:\n")
print(df_jobs.to_string(index=False))  # Print without index

# **STEP 8: Save to Excel**
excel_filename = "job_postings.xlsx"
df_jobs.to_excel(excel_filename, index=False)

# **STEP 9: Confirm Completion**
print(f"\nJob postings for {len(technologies)} technologies successfully saved in {excel_filename}!")
```

```
Job Postings by Technology:

  Technology  Job Postings
           C         25114
          C#           526
         C++           506
        Java          3428
  JavaScript          2248
      Python          1173
       Scala           138
      Oracle           899
  SQL Server           423
MySQL Server             0
  PostgreSQL            86
     MongoDB           208

Job postings for 12 technologies successfully saved in job_postings.xlsx!
```

In [34]:

```
# **STEP 7.1: Visualize the Results**
import matplotlib.pyplot as plt

# Sort by job postings (optional, for cleaner visualization)
df_jobs_sorted = df_jobs[df_jobs["Job Postings"] != "API Error"].copy()
df_jobs_sorted["Job Postings"] = df_jobs_sorted["Job Postings"].astype(int)
df_jobs_sorted = df_jobs_sorted.sort_values(by="Job Postings", ascending=False)

# Create a bar chart
plt.figure(figsize=(12, 6))
plt.bar(df_jobs_sorted["Technology"], df_jobs_sorted["Job Postings"])
plt.title("Number of Job Postings by Technology")
plt.xlabel("Technology")
plt.ylabel("Number of Job Postings")
plt.xticks(rotation=45)
plt.tight_layout()

# Show the chart
plt.show()
```



In [ ]:

In [39]:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Define technologies to search for
technologies = [
    "C", "C#", "C++", "Java", "JavaScript", "Python",
    "Scala", "Oracle", "SQL Server", "MySQL Server", "PostgreSQL", "MongoDB"
]

# Step 2: API URL
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# Step 3: Fetch data from API
response = requests.get(api_url)
if response.status_code != 200:
    print("Failed to fetch data from the API.")
else:
    job_data = response.json()

    # Step 4: Count job postings for each technology
    job_results = []
    for tech in technologies:
        count = sum(1 for job in job_data if tech.lower() in job.get("Key Skills", "").lower())
        job_results.append({"Technology": tech, "Job Postings": count})

    # Step 5: Create DataFrame
    df_jobs = pd.DataFrame(job_results)
    df_jobs = df_jobs.sort_values(by="Job Postings", ascending=False)

    # Step 6: Save to Excel
    excel_filename = "job-posting3.xlsx"
    df_jobs.to_excel(excel_filename, index=False)
    print(f"Excel file '{excel_filename}' created successfully.")

    # Step 7: Plot beautified bar chart
    plt.figure(figsize=(12, 6))
    colors = plt.cm.plasma(np.linspace(0, 1, len(df_jobs)))
    bars = plt.bar(df_jobs["Technology"], df_jobs["Job Postings"], color=colors)

    # Add value labels
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height + 1, str(height),
                 ha='center', va='bottom', fontsize=9, fontweight='bold')

    # Customize chart
    plt.title("Job Postings by Programming Language / Technology", fontsize=14, fontweight='bold')
    plt.xlabel("Technology")
    plt.ylabel("Number of Job Postings")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

    # Step 8: Save chart
    chart_filename = "job_postings_tech_chart3.png"
    plt.savefig(chart_filename, dpi=300)
    plt.show()

    print(f"Chart saved as '{chart_filename}' — ready for PowerPoint or report.")
```

Excel file 'job-posting3.xlsx' created successfully.

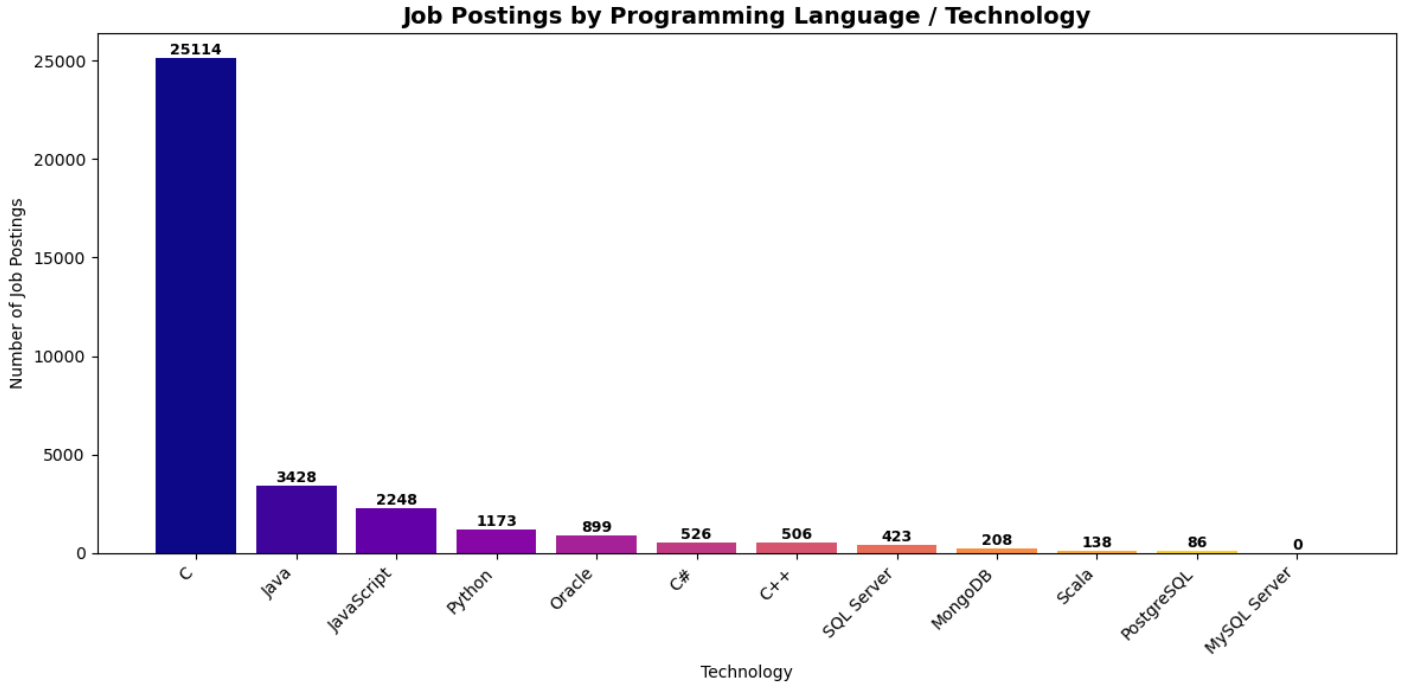


Chart saved as 'job_postings_tech_chart3.png' — ready for PowerPoint or report.

In [ ]:

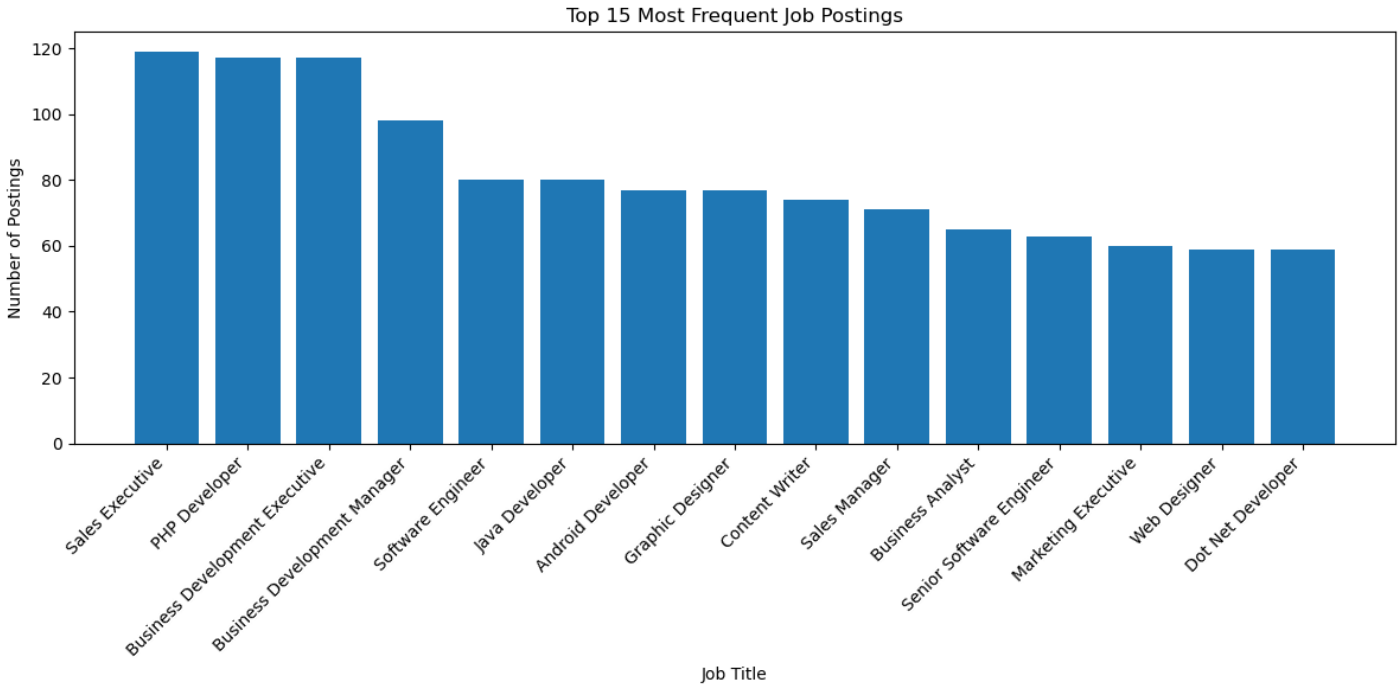
In [41]:

```
import requests
import pandas as pd
from collections import Counter

# API URL
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# Fetch data
response = requests.get(api_url)

if response.status_code == 200:
    job_data = response.json()

    # Extract all job titles
    job_titles = [job.get("Job Title", "Unknown") for job in job_data]

    # Count job title frequencies
    title_counts = Counter(job_titles)

    # Convert to DataFrame
    df_top_titles = pd.DataFrame(title_counts.items(), columns=["Job Title", "Job Postings"])
    df_top_titles = df_top_titles.sort_values(by="Job Postings", ascending=False).head(15)

    # Save to Excel
    excel_filename = "job-postings.xlsx"
    df_top_titles.to_excel(excel_filename, index=False)

    print(f"Excel file '{excel_filename}' created with top 15 job titles.")

else:
    print("Failed to fetch job postings from the API.")
```

Excel file 'job-postings.xlsx' created with top 15 job titles.

In [43]:

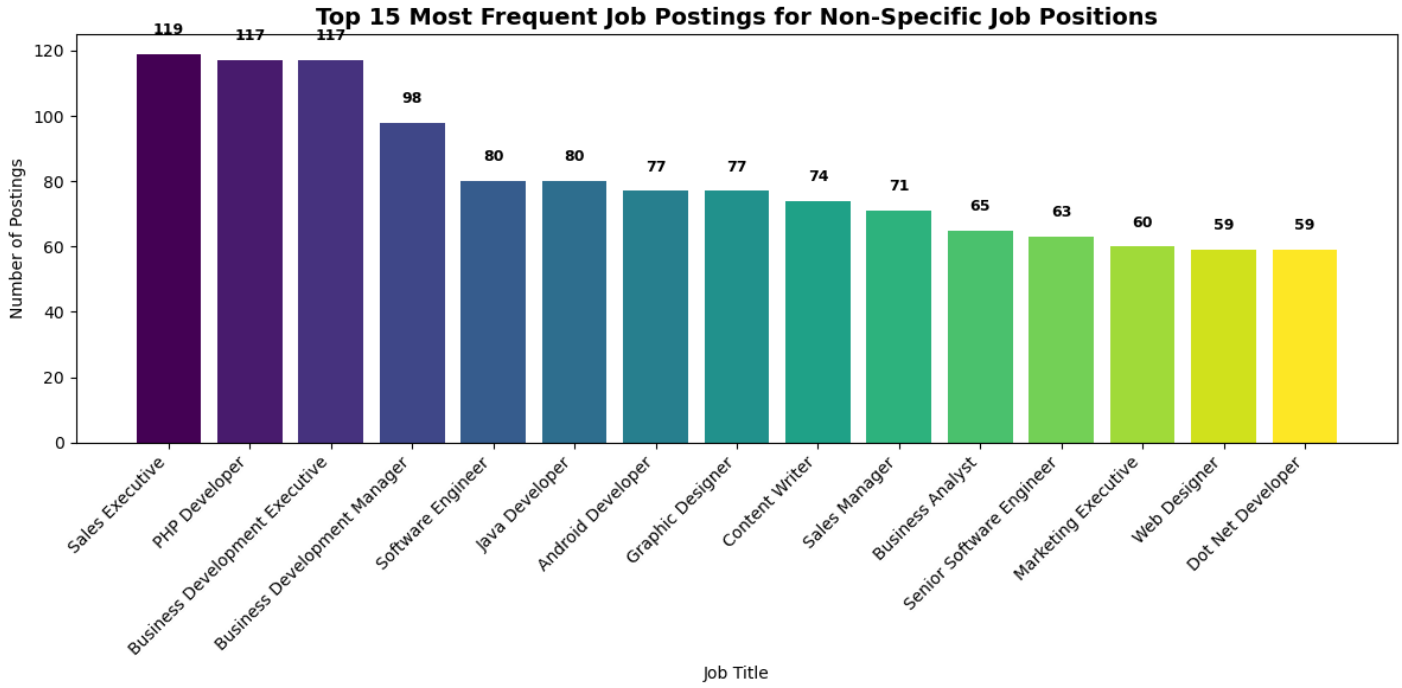
```
import matplotlib.pyplot as plt

# Load the Excel file
df_chart = pd.read_excel("job-postings.xlsx")

# Plot bar chart
plt.figure(figsize=(12, 6))
plt.bar(df_chart["Job Title"], df_chart["Job Postings"])
plt.title("Top 15 Most Frequent Job Postings")
plt.xlabel("Job Title")
plt.ylabel("Number of Postings")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Save as PNG image
chart_filename = "top_job_postings_chart.png"
plt.savefig(chart_filename, dpi=300)
plt.show()

print(f"Bar chart saved as '{chart_filename}' for use in PowerPoint.")
```




```
Bar chart saved as 'top_job_postings_chart.png' for use in PowerPoint.
```

In [45]:

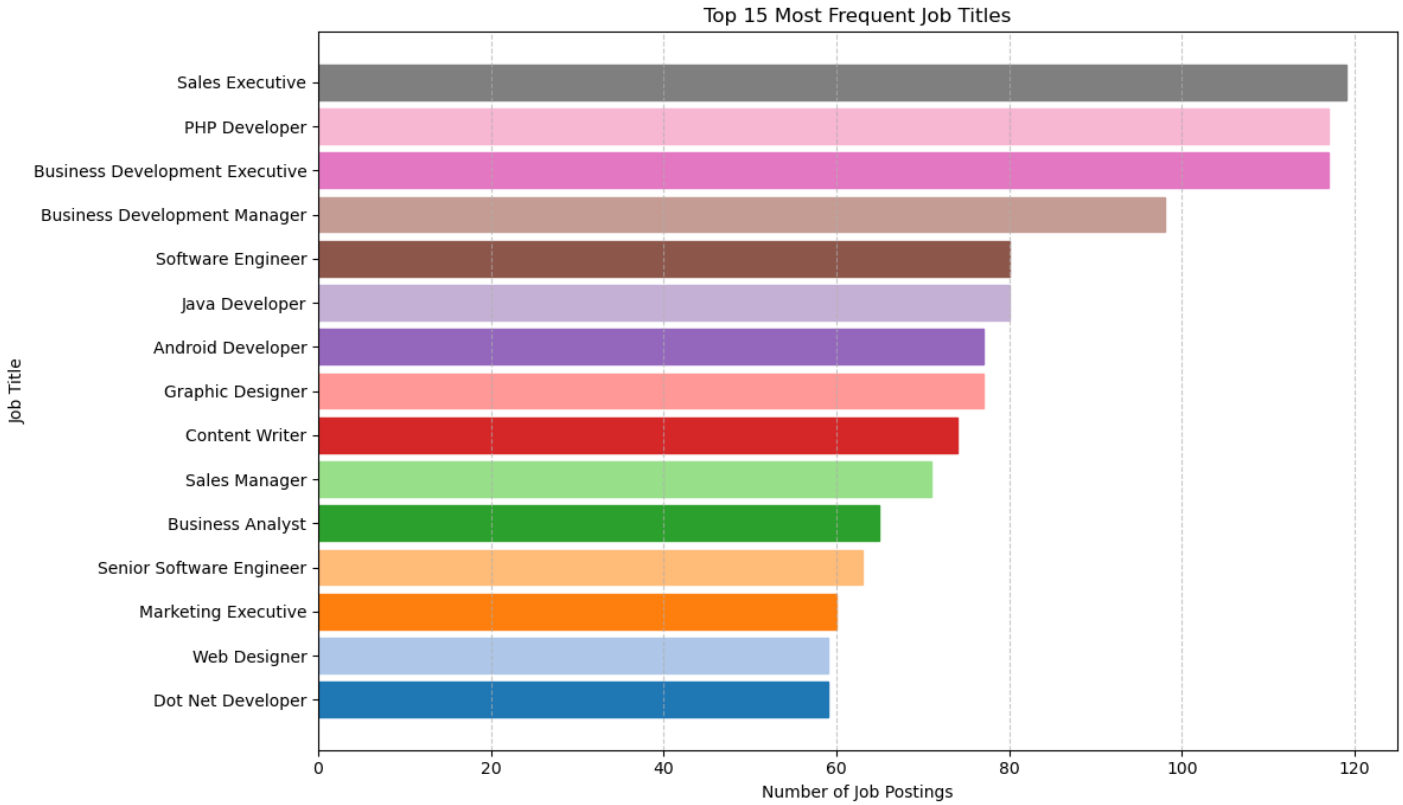
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Load the Excel file
df_chart = pd.read_excel("job-postings.xlsx")

# Step 2: Plot the bar chart with custom colors and labels
plt.figure(figsize=(12, 6))

# Create color palette (one color per bar)
colors = plt.cm.viridis(np.linspace(0, 1, len(df_chart)))  # You can change colormap: 'plasma', 'tab20', etc.

# Create bars
bars = plt.bar(df_chart["Job Title"], df_chart["Job Postings"], color=colors)

# Add value labels on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 5, str(height),
             ha='center', va='bottom', fontsize=9, fontweight='bold')

# Customize plot
plt.title("Top 15 Most Frequent Job Postings for Non-Specific Job Positions", fontsize=14, fontweight='bold')
plt.xlabel("Job Title")
plt.ylabel("Number of Postings")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Save as image for PowerPoint
chart_filename = "top_job_postings_chart_colored.png"
plt.savefig(chart_filename, dpi=300)
plt.show()

print(f"Beautified bar chart saved as '{chart_filename}' – ready for your PowerPoint slide!")
```




```
Beautified bar chart saved as 'top_job_postings_chart_colored.png' – ready for your PowerPoint slide!
```

In [89]:

```
# Data: Top 15 most frequent job titles
job_titles = [
    "Sales Executive", "PHP Developer", "Business Development Executive",
    "Business Development Manager", "Software Engineer", "Java Developer",
    "Android Developer", "Graphic Designer", "Content Writer", "Sales Manager",
    "Business Analyst", "Senior Software Engineer", "Marketing Executive",
    "Web Designer", "Dot Net Developer"
]

job_postings = [119, 117, 117, 98, 80, 80, 77, 77, 74, 71, 65, 63, 60, 59, 59]

# Create colorful bar chart
plt.figure(figsize=(12, 7))
bars = plt.barh(job_titles[::-1], job_postings[::-1])  # Reverse to have highest on top

# Color each bar differently
for i, bar in enumerate(bars):
    bar.set_color(plt.cm.tab20(i % 20))

# Add chart details
plt.title("Top 15 Most Frequent Job Titles")
plt.xlabel("Number of Job Postings")
plt.ylabel("Job Title")
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
# plt.show()

# Save figure
chart_path = "top_15_job_titles_chart.png"
plt.savefig(chart_path)
plt.show()

print(f"Top 15 Most Frequent Job Titles bar chart saved as '{chart_path}' – ready for your PowerPoint slide!")
```




```
Top 15 Most Frequent Job Titles bar chart saved as 'top_15_job_titles_chart.png' – ready for your PowerPoint slide!
```

In [47]:

```
# Show top 15
print("\nTop 15 most frequent job titles:\n")
print(df_top_titles.head(15))

 #df_top_titles = df_top_titles.sort_values(by="Job Postings", ascending=False).head(15)
```

Top 15 most frequent job titles:

	Job Title	Job Postings
68	Sales Executive	119
76	PHP Developer	117
86	Business Development Executive	117
117	Business Development Manager	98
766	Software Engineer	80
82	Java Developer	80
45	Android Developer	77
85	Graphic Designer	77
263	Content Writer	74
125	Sales Manager	71
659	Business Analyst	65
532	Senior Software Engineer	63
259	Marketing Executive	60
138	Web Designer	59
89	Dot Net Developer	59

Top 15 Most Frequent Job Titles

This horizontal bar chart highlights the job titles with the highest number of postings:

Key Insights

1. **Sales roles** (Sales Executive, Business Development Executive, Business Development Manager) dominate the market.
2. **Software development** and **technical roles** (PHP Developer, Software Engineer, Java Developer) are also highly demanded.
3. **Creative positions** (Graphic Designer, Content Writer, Web Designer) show notable hiring needs.

The data reflects strong demand for both **sales-oriented** and **technical IT skills** in the current job market.

In [50]:

```
import requests
import pandas as pd

# API URL containing job postings
api_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/module%201/Accessing%2(

# Fetch the data
response = requests.get(api_url)

if response.status_code == 200:
    job_data = response.json()

    # Extract all job titles using the correct key
    job_titles = [job.get("Job Title", "Unknown") for job in job_data]

    # Get unique job titles
    unique_titles = set(job_titles)

    # Convert to DataFrame for better display
    df_unique_titles = pd.DataFrame(unique_titles, columns=["Unique Job Titles"])

    # Print number of unique job postings and display a few samples
    print(f"Total unique job postings: {len(unique_titles)}")
    print("\nSample of job titles:\n")
    print(df_unique_titles.head(10))  # Show more if you want
else:
    print("Failed to fetch job postings from the API.")
```

Total unique job postings: 21644

Sample of job titles:

	Unique Job Titles
0	International Outbound Voice Process / UK Shift
1	SAP S/4hana Project Systems
2	Opening For Junior Design Engineer (electrical)
3	Network Engineer
4	Senior Manager - Compliance Change Management
5	Mechanical Engineer Fresher- Chennai & Coimbatore
6	Customer Care Executive (Inbound Tech Voice P...
7	Travel Sales Executive for Travel Company in .
8	Creative Visualiser
9	Credit Head :: Co. op Bank :: Ahmedabad

In [52]:

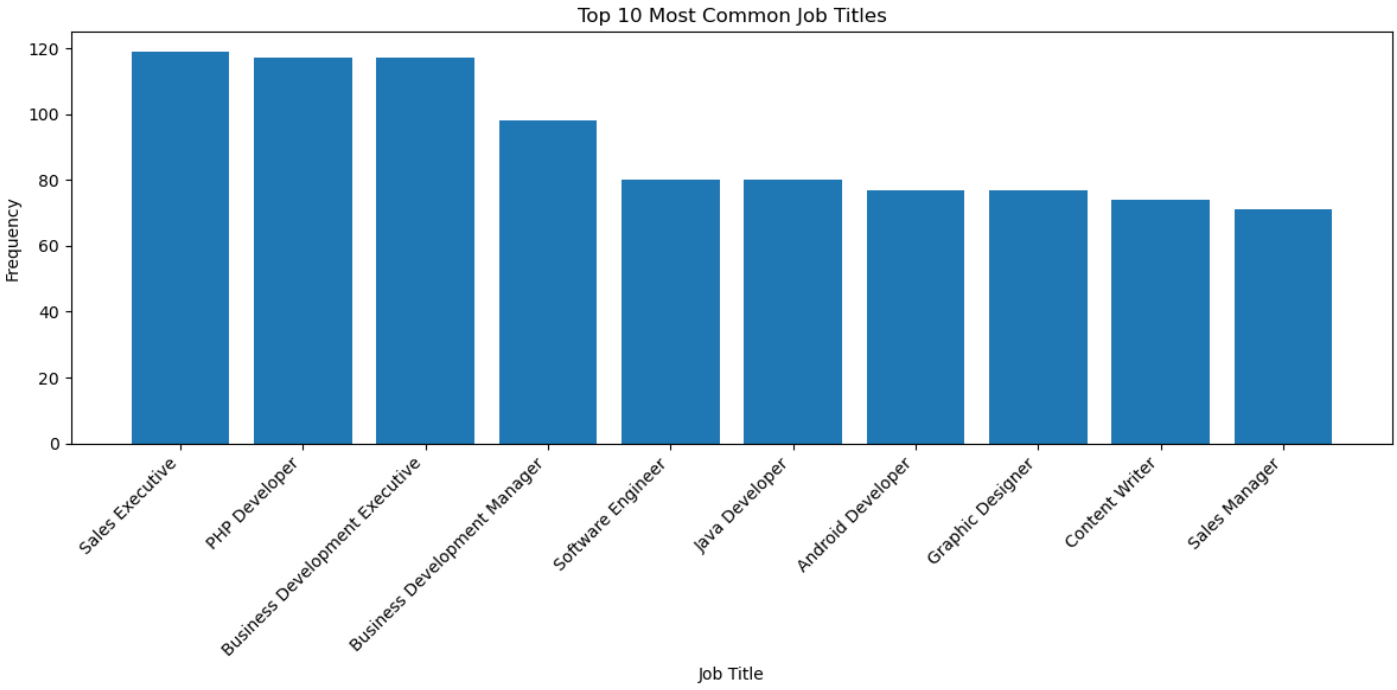
```
from collections import Counter

# Count occurrences of each job title
title_counts = Counter([job.get("Job Title", "Unknown") for job in job_data])

# Convert to a DataFrame and sort
df_title_counts = pd.DataFrame(title_counts.items(), columns=["Job Title", "Count"])
df_title_counts = df_title_counts.sort_values(by="Count", ascending=False)

# Show top 10
print("\nTop 10 most frequent job titles:\n")
print(df_title_counts.head(10))
```

Top 10 most frequent job titles:

	Job Title	Count
68	Sales Executive	119
76	PHP Developer	117
86	Business Development Executive	117
117	Business Development Manager	98
766	Software Engineer	80
82	Java Developer	80
45	Android Developer	77
85	Graphic Designer	77
263	Content Writer	74
125	Sales Manager	71

In [54]:

```
import matplotlib.pyplot as plt

# Plot top 10 job titles
top_n = 10
top_titles = df_title_counts.head(top_n)

plt.figure(figsize=(12, 6))
plt.bar(top_titles["Job Title"], top_titles["Count"])
plt.title(f"Top {top_n} Most Common Job Titles")
plt.xlabel("Job Title")
plt.ylabel("Frequency")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [56]:
```
import os

# Get the absolute file path of the notebook file
file_path = os.path.abspath("Collecting_job_data_using_APIs-Lab.ipynb")
print("The notebook is located at:", file_path)
```

```
The notebook is located at: C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-using-APIs\Collecting_job_data_using_APIs-Lab.ipynb
```

In [58]:
```
import os

# Get the current working directory
file_path = os.path.abspath("Collecting_job_data_using_APIs-Lab.ipynb")

print(f"The file is located at: {file_path}")
```

```
The file is located at: C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-using-APIs\Collecting_job_data_using_APIs-Lab.ipynb
```

In [60]:
```
from pathlib import Path

# Get the absolute path
file_path = Path("Collecting_job_data_using_APIs-Lab.ipynb").resolve()

print(f"The file is located at: {file_path}")
```

```
The file is located at: C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-using-APIs\Collecting_job_data_using_APIs-Lab.ipynb
```

In [62]:
```
for file in os.listdir():
    if file.endswith(".ipynb"):
        print(f"Notebook Found: {os.path.abspath(file)}")
```

```
Notebook Found: C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-using-APIs\Collecting_job_data_using_APIs-Lab.ipynb
```

In [64]:
```
!pip install pdfkit
```

```
Requirement already satisfied: pdfkit in c:\users\ede\anaconda3\lib\site-packages (1.0.0)
```

In [ ]:

In [66]:

```
import nbconvert
import nbformat
import pdfkit

# Corrected file paths (Using raw string notation or forward slashes)
input_file_path = r"C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-usi
output_pdf_path = r"C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-u

# Load the Jupyter Notebook file
with open(input_file_path, 'r', encoding='utf-8') as f:
    notebook_content = nbformat.read(f, as_version=4)

# Convert the notebook to HTML
html_exporter = nbconvert.HTMLExporter()
html_exporter.exclude_input = False  # Include code cells in the output
(body, resources) = html_exporter.from_notebook_node(notebook_content)

# Convert HTML to PDF
pdfkit.from_string(body, output_pdf_path)

# Return the PDF file path
print(f"Notebook successfully converted to PDF: {output_pdf_path}")
```

```
Notebook successfully converted to PDF: C:\Users\Ede\Desktop\IBM_Capstone_Data_Analyst_2025\Module_1_Real_World_Projects\Project_1_Data-Collection-u
sing-APIs\Collecting_job_data_using_APIs-Lab.pdf
```

In [96]:

```
!jupyter nbconvert --to html "Collecting_job_data_using_APIs-Lab.ipynb"
```

```
[NbConvertApp] Converting notebook Collecting_job_data_using_APIs-Lab.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 10 image(s).
[NbConvertApp] Writing 1120184 bytes to Collecting_job_data_using_APIs-Lab.html
```

Congratulations to us for having successfully completed the above lab!

Authors:

Kelechukwu Innocent Ede and Ayushi Jain

Other Contributors:

- Rav Ahuja
- Lakshmi Holla
- Malika

In [ ]: